.Net Application Domains

Jim Fawcett

CSE681 – Software Modeling and Analysis

Fall 2015

Agenda

- Role of AppDomains
  - Application isolation
    - Visibility
    - Data
    - Security settings
  - Type safety and verification
  - Dynamic application extensions
- AppDomain structure
- AppDomain managers
- Summary

References

- Common Language Runtime, Steven Pratschner, Microsoft Press, 2005
- Essential .Net, Volume 1, Don Box with Chris Sells, Addison-Wesley, 2003
- www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/TestHarnessPrototype

Role of Application Domains

- Memory Access Isolation
  - Isolate unrelated applications from each other at run time.
    - Win32 Processes
    - CLR Appdomains
  - Intent is to make system as stable as possible and minimize security exploits.
    - Web services and ASP applications run in AppDomains to isolate them from IIS

What is an Application Domain?

- An execution environment in which managed code runs
- Safe managed code in an application domain is isolated from safe managed code running in any other.
  - Safe code is code that can be verified by the JIT compiler.
  - C# with no unsafe regions, is safe code.
- Application domains are cheaper to start, unload, and run, than Windows processes, in terms of CPU cycles and memory.
- It is cheaper to make calls between AppDomains than between Windows processes.

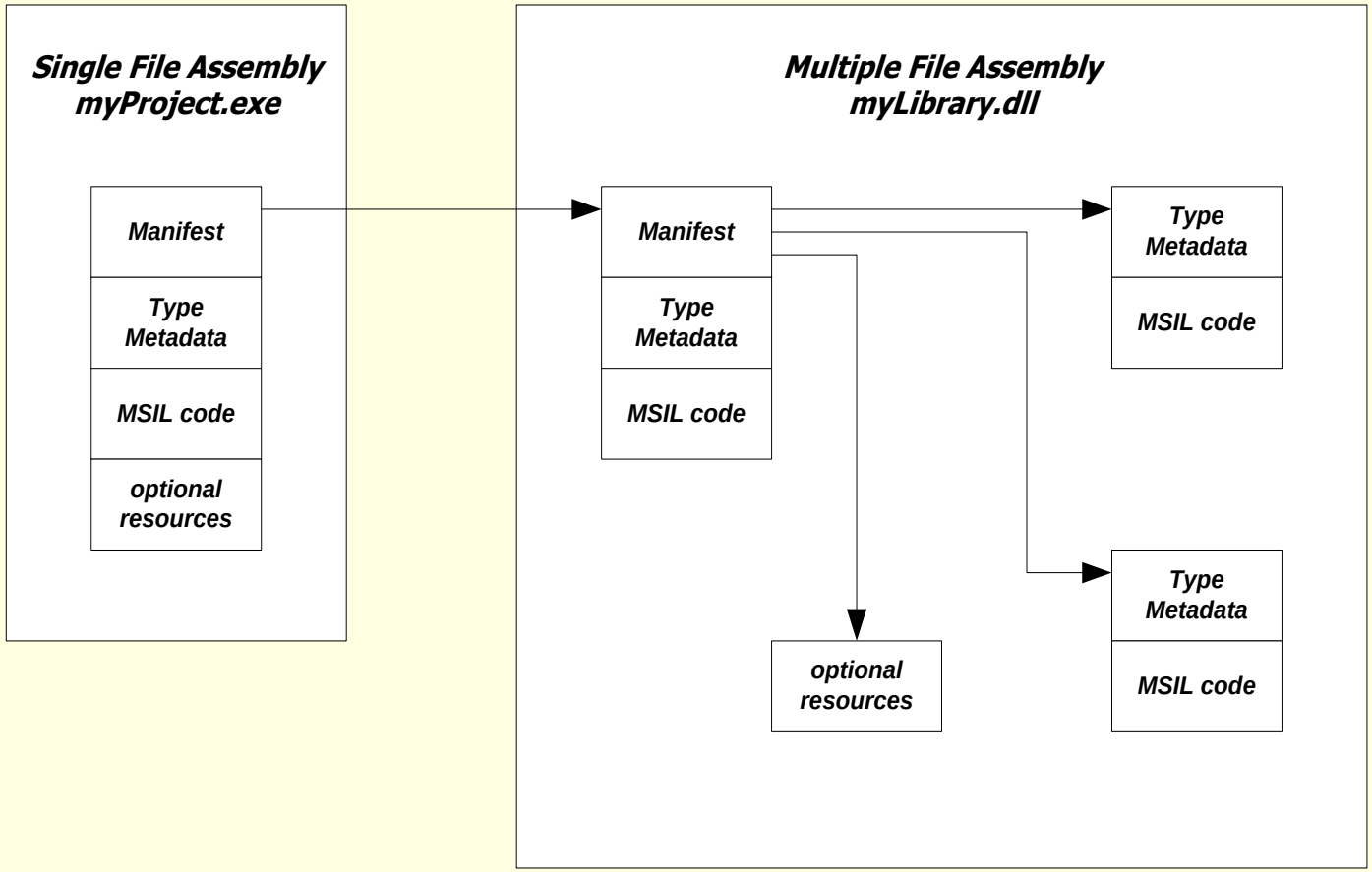
What is a Module?

- CLR programs reside in Modules
- Modules contain:
  - Code(IL code)
  - MetaData (module description)
  - Resources (any external resource)
- Modules are not deployable components

Assemblies

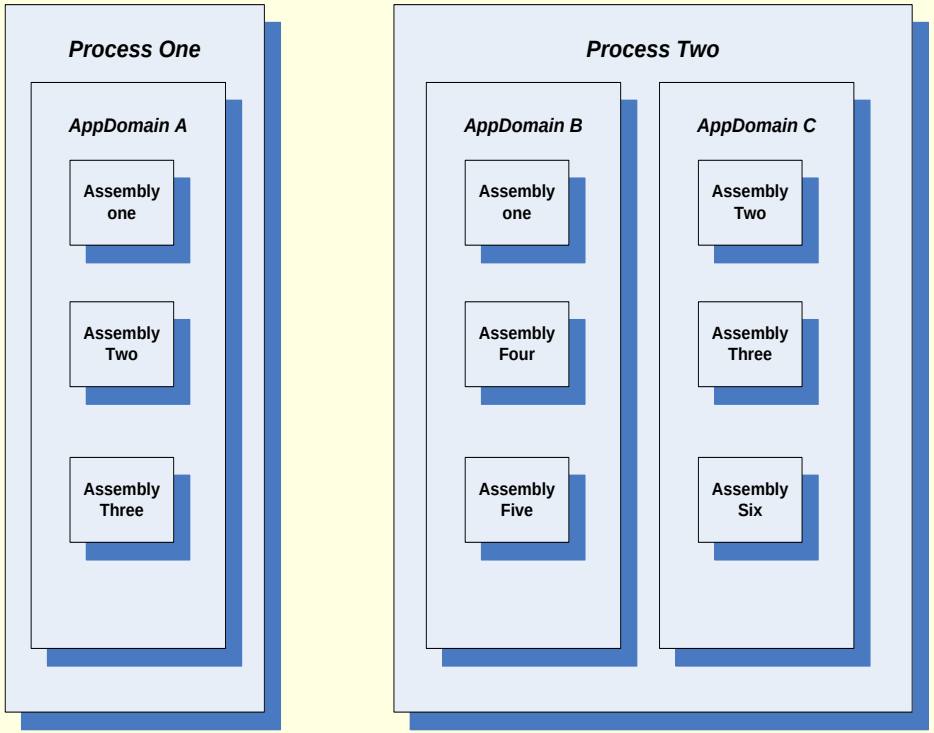
- Modules are the physical structure of a program that resides in file system
- Assembly is a logical construct that the CLR uses to access modules
- Assemblies are deployable modules
  - Each assembly has a manifest
  - Assemblies might have multiple modules
  - Only one manifest exists per assembly
- Manifests describe the modules in the assembly
- Assembly can be:
  - Executable application
  - Library

Assemblies


Single File Assembly
myProject.exe
Manifest
Type Metadata
MSIL code
optional resources
Multiple File Assembly
myLibrary.dll
Manifest
Type Metadata
MSIL code
optional resources
Type Metadata
MSIL code
Type Metadata
MSIL code

Application Domains and Processes

- Each application domain runs in the context of one and only one native Windows process.
- A windows process can have no application domains, one, or many.



Creating Child AppDomains

- AppDomainSetup dInfo
  = new AppDomainSetup();

  dInfo.ApplicationName = ADname;

  Evidence evidence
  = AppDomain.CurrentDomain.Evidence;

- AppDomain child
  = AppDomain.CreateDomain(
  ADname, evidence, dInfo
  );

Loading Assemblies into AppDomain

- If child is a child AppDomain, then:
  - child.Load(assembly);

    Probes paths beneath application and private paths to find an assembly to load, using Fusion rules, where assembly is the assembly name, without extension.
    - Load can be called by anyone with a reference to the AppDomain instance.

  - Assembly.LoadFrom(fileSpec);

    Loads specific assembly into current AppDomain, so to load into the child, this must be called from code in the child domain.
    - This is what the Test Harness prototype does.

Unloading

- The Win32 API provides LoadLibrary and UnloadLibrary for injecting and removing libraries from an application dynamically.
- The .Net CLR does support loading, but does not support dynamically unloading libraries.
- You have to create a child domain, load libraries into it, and unload the domain when you are done, using:

  public static AppDomain.Unload(AppDomain);

Communicating between AppDomains

- Creating and using types in child domain:
  - ObjectHandle oh
    = ad.CreateInstance(Assembly, aType);
  - aType p = oh.Unwrap() as aType;
  - Use p, a proxy, just like an instance of aType.

This creates a proxy, typed by the CLR as aType. The proxy, p, marshals all calls to the real object in the child domain.

Communicating between AppDomains

- Access using AppDomain Dictionary
  - public virtual void AppDomain.SetData(string key, object value);

    marshals a reference into dictionary.
  - public virtual object AppDomain.GetData(string key);

    returns a proxy to object in other domain.
  - Dictionary objects must derive from MarshalByRefObject

Isolation – Win32 IIS Example

- In IIS, Prior to .Net, you had the choices:
  - Load and run (**ISAPI**) application dlls in IIS process and possibly take down the server.
  - Run (**CGI**) application as a separate process, paying interprocess communication performance penalty.
  - Run (**ASP**) script with scripting performance penalty and development issues.
  - Use "standard" **COM** objects loaded inproc and accessed from script to improve performance, so just like ISAPI, but known quantities "may" be safe.

Isolation - .Net IIS Example

- In IIS, with .Net, you have all the previous choices plus:
  - Run applications (**ASP.Net** and **Web Services**) each in its own child AppDomain, loaded by IIS, but isolated from it.
    - CLR isolates code loaded into child domain from the application running in primary AppDomain.
    - This is the default processing model supported by both ASP.Net and Web Services.

Objects and Types

- An object resides in exactly one AppDomain, as do values.
- Object references must refer to objects in the same AppDomain.
- Like objects, types reside in exactly one AppDomain. So if two AppDomains need to use a type, one must initialize and allocate the type once per AppDomain

Types cont.

- If a type is used in more than one AppDomain, one must load and initialize the type's module and assembly once for each AppDomain the type is used in.
- Since each such AppDomain maintains a separate copy of the type, each has its own private copy of the type's static fields.

Isolation – Visibility

- Type visibility
  - When a type is loaded into a child AppDomain it is visible only within that domain unless it is also loaded or marshaled back into the primary domain.
    - An instance of a type can be marshaled by value, which results in a serialization, transmission, and deserialization.
      - Class must be attributed as [serializable()]
    - An instance of a type can also be marshaled by reference, which creates a proxy in the using domain.
      - Class must derive from MarshalByRefObject
    - Usually, we want to marshal by reference, because we want the instance to run in the child domain.

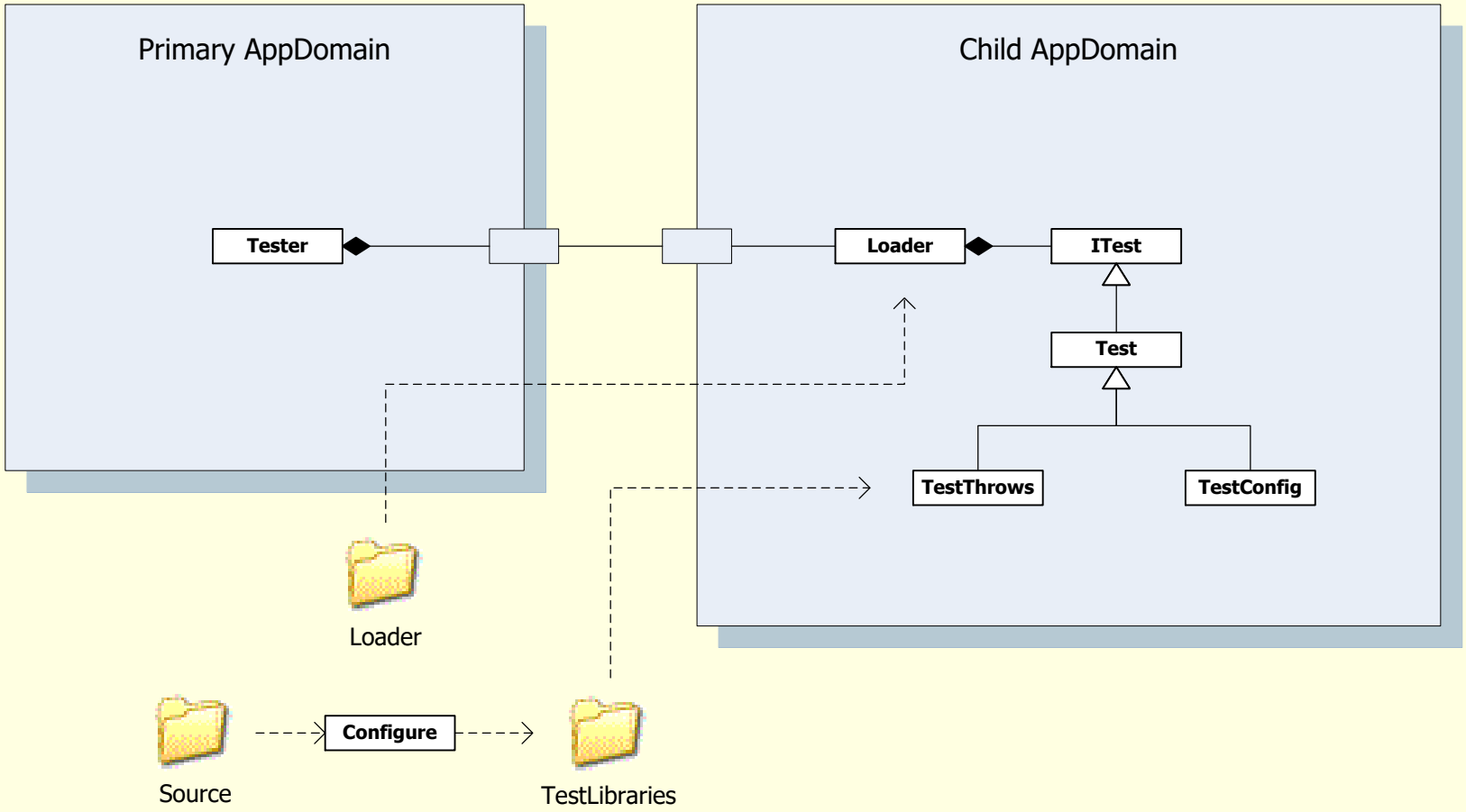
Resources and Memory

- An AppDomains' resources are held in memory as long as the owning AppDomain is loaded.
- Unloading an AppDomain is the only way to unload a module or an assembly or to reclaim the memory consumed by a type's static fields.

Test Harness Example

- The test harness example, discussed in CSE681 and CSE784 illustrates this visibility:
  - The primary domain coerces a child domain to load a "loader" into a child domain and marshal back a reference to it.
  - The primary domain then, using the loader reference, instructs it to load a collection of test assemblies for processing.
  - The affects of this are:
    - The primary AppDomain only knows about the loader type, not all the testing types.
    - The test manager, running in the Primary AppDomain is isolated from failures of the test and tested code.

Test Harness Configuration



Isolation – Configuration Data

- Each AppDomain in a process can be independently configured, either programmatically or with a configuration file.
  - Each application domain may have a configuration file that can be used to customize:
    - Local search paths
    - Versioning policy, e.g., what is allowed to run
    - Remoting information
    - User defined settings

AppDomain Config Files

- An AppDomain config file resides in the process exe's directory and has the process exe's name with .config extension:
  - myProcess.exe.config
- That can be changed with AppDomainSetup.ConfigurationFile = newPath;
- Some examples:
  http://blogs.msdn.com/suzcook/archive/2004/05/14/132022.aspx
- Config file schema:
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfNETFrameworkConfigurationFileSchema.asp

Isolation – Security Settings

- Application domains can be used to modify Code Access Security (CAS) settings applied to code running within the domain.
  - You can modify CAS policy for the domain. That maps code identity, based on evidence, to a set of granted permissions.
  - You can also set security evidence on the domain itself. If the grants for the domain are less than the grants for the assembly, the domain wins, and vice versa.

Isolation - Static Data

- Static members of classes are isolated by AppDomains:
  - If the same type is loaded into a parent and child domain, they are considered to be distinct types, and do not share static members.
  - If code is loaded domain-neutral, the code base is shared, but separate copies are maintained for all static members.

Process Resources not Isolated

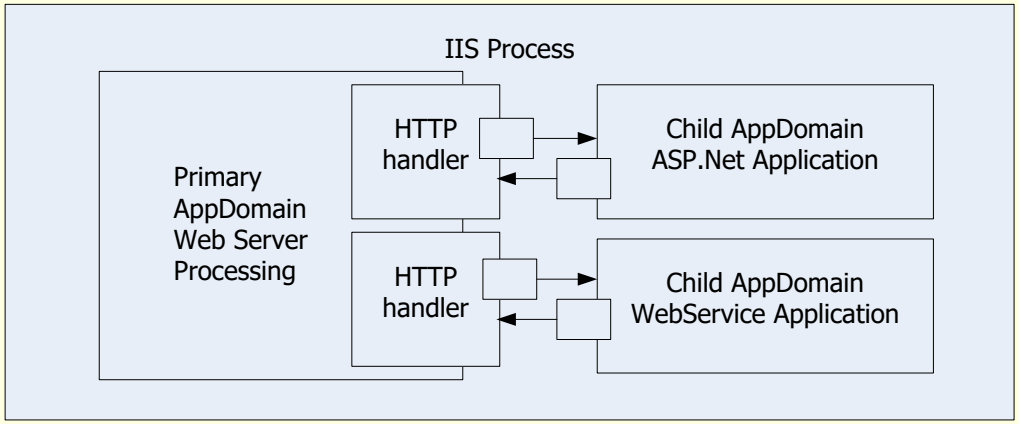
- Resources not isolated to an AppDomain:
  - Managed heap
  - Managed threads
    - CLR prevents data and behavior leaks
  - Managed ThreadPool
  - Mutexes and Events
    - If named, these kernel objects are shared across AppDomains

AppDomain Events

- The AppDomain Type supports a handful of events that allow interested parties to be notified of significant conditions in a running program.
- Events:
  - AssemblyLoad
  - AssemblyResolve
  - TypeResolve
  - ResourceResolve
  - DomainUnload
  - ProcessExit
  - Unhandled Exception

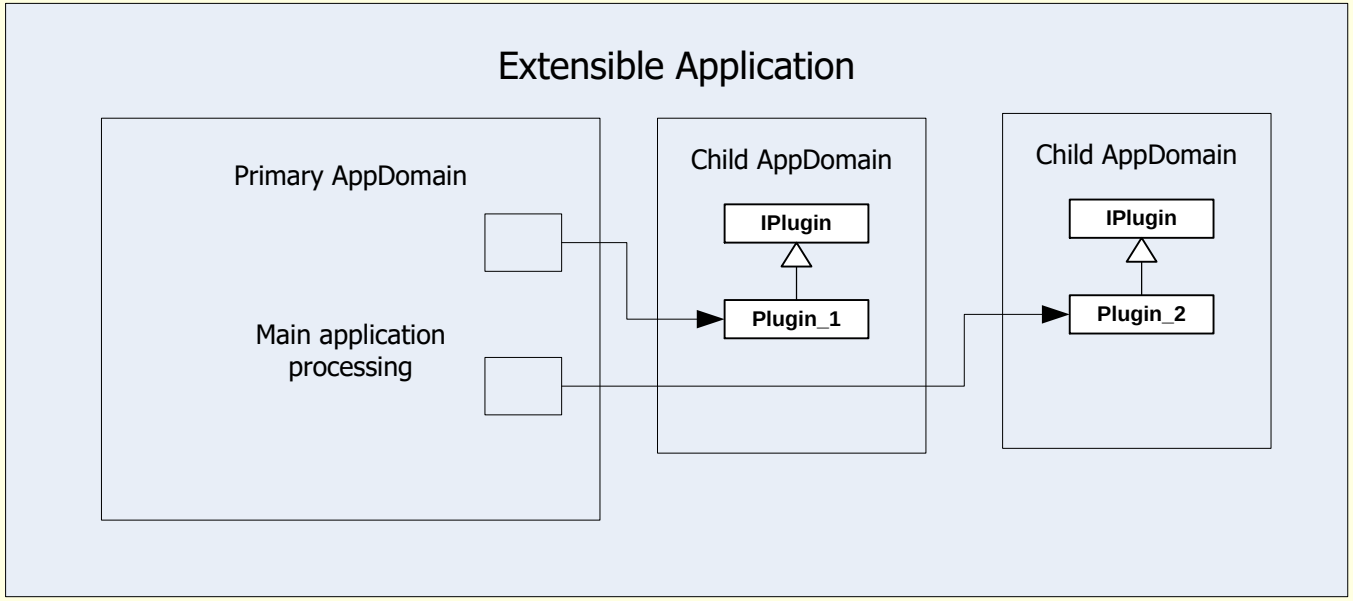
IIS Application Domain Structure

- When application starts by getting first request, IIS creates a child domain, loads the application into it.
- Request details are extracted and processed by HTTP handler. Handler creates and uses instance of application.



Plugin Architecture Structure

- Load plugin assemblies
- Use reflection to find plugin types and ensure that they implement IPlugin.
- Create and use type as shown earlier.



AppDomain Managers

- AppDomain Managers are available in .Net version 2.
  - The System namespace provides a base definition, which your applications will specialize.
  - Looks like they are intended to do about what my Loader does.

AppDomain Managers

- The CLR loads your AppDomain manager into each application domain created in the process.
- The manager intercepts all calls to CreateDomain, allowing you to configure the domain as needed by the application.
- AppDomainManager.ApplicationActivator(…) activates plugins defined by a "formal" manifest.
- You define the AppDomain manager to a process using either CLR hosting APIs (COM) or using a set of Environment variables using a configuration file.

Summary

- AppDomains provide:
  - Isolation
  - Control of type visibility
  - Fine-grained configuration of loading, versioning, remoting, user settings
  - Programmatic control of Library loading and unloading
  - Marshaling services to access type instances in another domain.
- AppDomains are used by:
  - ASP.Net, Web Services, IExplorer, …

End of Presentation

Appendix – Dynamic Code Generation

- This material was developed by Vijay Appurdai, as a presentation for our Brown-Bag Seminar series.
  - His primary source was "Essential .Net", Don Box and Chris Sells, Addison-Wesley, 2003

AppDomains and Assembly Resolver

- AppDomains play a critical role in controlling the behavior of the assembly resolver
- Each AppDomain can have its own APPBASE and configuration file. So each can have its own probe path and version policy
- The AppDomain stores the properties used by the assembly resolver in a data structure called AppDomainSetup which is maintained on a per-AppDomain basis.

AppDomains and Dynamic Directories

- Consider the case in which an application needs to generate code dynamically.
- If the application needs to load the code by probing, then the application needs to have write access to a directory underneath APPBASE
- However we may want to execute code from a read-only part of file system.

Dynamic Directories

- This means that we need to have an alternate location for dynamic code generation.
- This is the role of the AppDomain.DynamicDirectory property.
- Each AppDomain may have at most one dynamic directory.
- This dynamic directory is added automatically to the probe path. ASP.Net is a heavy user of this feature

Shadow Copying

- Shadow copying addresses the problem related to server side development and deployment.
- The classic Win32 loader takes a read lock on a file that it loads to ensure that no changes are made to the underlying executable image.
- So overwriting this dll with a new version requires shutting down the server.

.Net Solution

- In .Net we have the shadow copying facility.
- When the CLR loads an assembly using shadow copying, a temporary copy of the underlying files is made in a different directory.
- These temporary files are loaded in lieu of the original assemblies.
- When shadow copying is enabled for an AppDomain, we need to specify two directory paths.

Shadow Copying cont.

- One path is the directory which needs to be shadow copied.
- The other is the path to which it needs to be shadow copied.
- This can be accomplished using the SetShadowCopyPath() and the SetCachePath() functions provided by the AppDomain class.
- Again, ASP.NET is a heavy user of this feature

AppDomains and Code Management

- Each AppDomain has its own private copy of a type's static data.
- The JIT compiler can generate code either on a per-AppDomain basis or on a per-process basis. So we can decide which one to use.
- There are three types of Loader Optmizations
  - SingleDomain
  - MultiDomain
  - MultiDomainHost

SingleDomain

- The SingleDomain assumes that the process will contain only one AppDomain.
- The JIT compiler therefore generates machine code seperately for each domain.
- This makes static field access faster and because we expect only one AppDomain we generate only one copy of machine code.

MultiDomain

- The MultiDomain flag assumes that the process contains several AppDomains running the same application.
- The JIT compiler generates only one machine code for the entire process.
- This makes static field access slower but significantly reduces memory needed.

MultiDomain Host

- This flag assumes that the process will contain several AppDomains, each of which will run different Application code.
- In this hybrid mode, only assemblies loaded from the GAC share machine code. (MultiDomain)
- Assemblies not loaded from GAC are assumed to be used only by the loading AppDomain.(SingleDomain)
- ASP.Net uses this flag.